**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МОЭВМ**

**КУРСОВАЯ РАБОТА**

**по дисциплине «Алгоритмы и структуры данных»**

**ТЕМА:** АВЛ - ДЕРЕВЬЯ

Вариант № 18

Студент гр. 8304 _____ Рыжиков А. В.

Преподаватель _____ Фирсов К. В.

Санкт-Петербург

2019

**ЗАДАНИЕ**
**НА КУРСОВУЮ РАБОТУ**

Студент Рыжиков Александр Владимирович

Группа 8304

Тема работы AVL-деревья

Исходные данные:
Написать программу создание программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов. Задания и ответы должны выводиться в файл в удобной форме: тексты заданий должны быть готовы для передачи студентам

Содержание пояснительной записки:

- Содержание
- Введение
- AVL-деревья
- Тестирование
- Исходный код
- Использованные источники

Дата выдачи задания: 11.10.2019
Дата сдачи реферата:

Дата защиты реферата:22.10.2019

Студент _____ Рыжиков А. В

Преподаватель _____ Фирсов К. В.

## АННОТАЦИЯ

В данной работе была создана программа на языке программирования C++, которая создаёт необходимое количество вариантов различной сложности и ответы к ним. Также было написано мобильное приложение под android на языке Java и с использованием базы данных Firebase. Приложение позволяет легко и быстро проводить тестирование студентов. При этом часть кода генерирующая задание и ответ написана на C++ и идентична коду, используемому в консольном приложении.

## SUMMARY

In this work, a program was created in the C++ programming language, which creates the necessary number of options of varying complexity and answers to them. Also, a mobile application for android was written in Java and using the Firebase database. The application allows you to easily and quickly test students. The part of the code that generates the task and response is written in C++ and is identical to the code used in the console application.

**СОДЕРЖАНИЕ**


| | | |
|---|---|---|
| | Введение | 5 |
| 1. | AVL-деревья | 6 |
| 2. | Описание структур данных и функций | 7 |
| 3. | Консольное приложение | 9 |
| 4. | Android приложение | 10 |
| 5. | Тестирование | 13 |
| | Заключение | 15 |
| | Список использованных источников | 17 |

# ВВЕДЕНИЕ

Целью данной курсовой работы является реализация программы для проверки построения AVL-деревьев. АВЛ-дерево – это сбалансированное двоичное дерево поиска, в котором поддерживается следующее свойство: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

АВЛ-деревья названы по первым буквам фамилий их изобретателей, Г. М. Адельсона-Вельского и Е. М. Ландиса, которые впервые предложили использовать АВЛ-деревья в 1962 году.

# 1. AVL-деревья

АВЛ-дерево (AVL tree) –сбалансированное по высоте двоичное дерево поиска, в котором у любой вершины высота левого и правого поддеревьев различаются не более чем на 1.

Если вставка или удаление элемента приводит к нарушению сбалансированности дерева, то выполняется его балансировка. Коэффициент сбалансированности узла (balance factor) – это разность высот его левого и правого поддеревьев. В АВЛ-дереве коэффициент сбалансированности любого узла принимает значения из множества {-1, 0, 1}.

После добавления нового элемента необходимо обновить коэффициенты сбалансированности родительских узлов. Если любой родительский узел принял значение -2 или 2, то необходимо выполнить балансировку поддерева путем поворота (rotation).

Типы поворотов:

1) Одиночный правый поворот (R-rotation, single right rotation)
2) Одиночный левый поворот (L-rotation, single left rotation)
3) Двойной лево-правый поворот (LR-rotation, double left-right rotation)
4) Двойной право-левый поворот (RL-rotation, double right-left rotation)

# 2 Описание структур данных и функций

```
struct Node {
    int key;
    unsigned char height;
    Node *left;
    Node *right;

    Node(int k) {
        key = k;
        left = right = 0;
        height = 1;
    }
};
```

Структура Node хранит  указатель на левое и правое поддерево, хранит в себе ключ и высоту.

## Функция insert.

Функция вставляет элемент в авл дерево в зависимости от его ключа и проводит балансировку дерева.

```
Node *insert(Node *p, int k) {
    if (!p) return new Node(k);
    if (k < p->key)
        p->left = insert(p->left, k);
    else
        p->right = insert(p->right, k);
    return balance(p);
}
```

## Функция balance.

Функция приводит дерево в сбалансированное состояние. В зависимости от коэффициента балансировки элементов  и нарушения сбалансированности применяются 4 выше описанные поворота.

```
const int right_overweight = 2;
const int left_overweight = -2;

Node *balance(Node *p) {
    fixHeight(p);
    if (bFactor(p) == right_overweight) {
        if (bFactor(p->right) < 0)
            p->right = rotateRight(p->right);
        return rotateLeft(p);
    }
    if (bFactor(p) == left_overweight) {
        if (bFactor(p->left) > 0)
            p->left = rotateLeft(p->left);
```

```
        return rotateRight(p);
    }
    return p;
}
```

# Функция левого правого поворота

```
Node *rotateRight(Node *p) {
    Node *q = p->left;
    p->left = q->right;
    q->right = p;
    fixHeight(p);
    fixHeight(q);
    return q;
}

Node *rotateLeft(Node *q) {
    Node *p = q->right;
    q->right = p->left;
    p->left = q;
    fixHeight(q);
    fixHeight(p);
    return p;
}
```

# Вспомогательные функции

Функция height возвращает глубину дерева.

Функция fixHeight устанавливает новое значение глубины.

Функция bFactor возвращает коэффициент сбалансированности.

```
unsigned char height(Node *p) {
    return p ? p->height : 0;
}

int bFactor(Node *p) {
    return height(p->right) - height(p->left);
}

void fixHeight(Node *p) {
    unsigned char hl = height(p->left);
    unsigned char hr = height(p->right);
    p->height = (hl > hr ? hl : hr) + 1;
}
```

# 3 Консольное приложение

1. Проверяющему предлагается ознакомится с тестами, находящимися в файле. Содержимое теста аналогично вводимым данным.

2. Пользователю предлагается выбрать уровень сложности и количество вариантов.

Сложность 1 – 5 элементов для построения и 1 для удаления

Сложность 2 – 10 элементов для построения и 2 для удаления

Сложность 3 – 15 элементов для построения и 3 для удаления

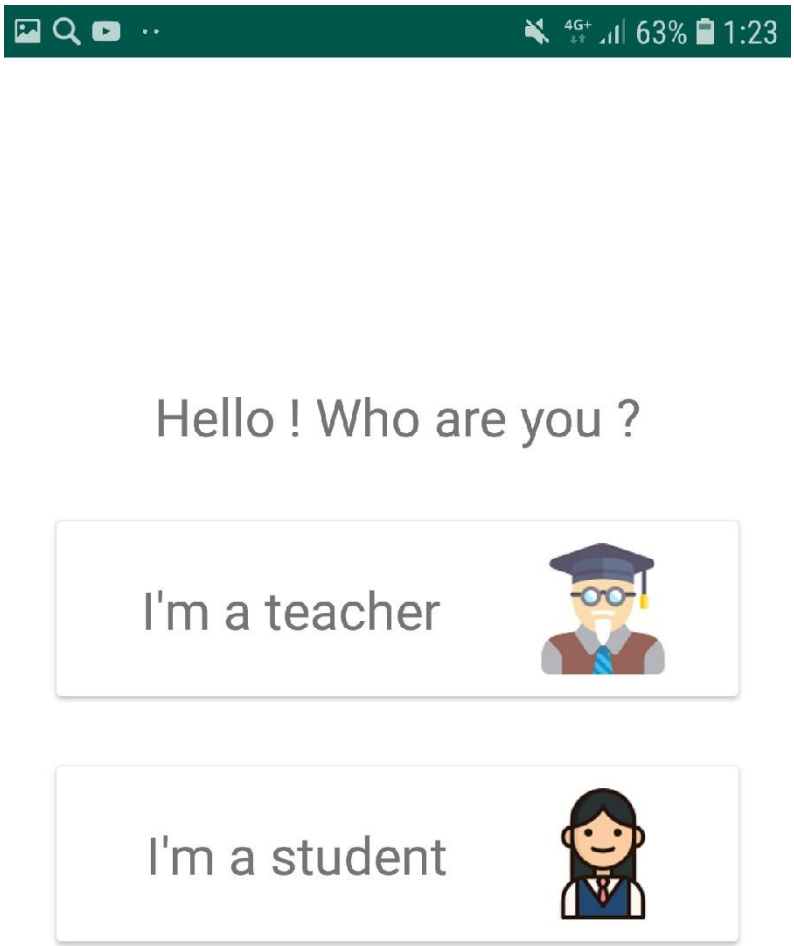
Сложность 4 – кастомный вариант  сложности, количество элементов для построения и для удаления вводятся вручную

```
Welcome to the verification program
The program creates variants of different and answers

If you want to see tests entre 'test', otherwise just any characters
no
Choose the difficulty
If you want to choose a simple difficulty - enter 1
If you want to choose a medium difficulty - enter 2
If you want to choose a high difficulty   - enter 3
If you want to choose a custom difficulty - enter 4
2
Enter the number of options (from 1 to 30)
5
Option number 1
      Insert: 8 3 7 6 1 2 4 5 9 10
      Delete: 5 7
      Answer: (3(1(2))(8(6(4))(9(10))))
Option number 2
      Insert: 9 6 2 5 1 3 4 7 8 10
      Delete: 8 3
      Answer: (5(2(1)(4))(7(6)(9(10))))
Option number 3
      Insert: 8 6 1 9 2 4 3 5 7 10
      Delete: 5 4
      Answer: (6(2(1)(3))(8(7)(9(10))))
Option number 4
      Insert: 3 6 7 1 5 10 2 4 8 9
      Delete: 7 10
      Answer: (6(3(1(2))(5(4)))(9(8)))
Option number 5
      Insert: 8 3 6 2 4 1 5 7 9 10
      Delete: 7 6
      Answer: (8(3(2(1))(4(5)))(9(10)))
If you want to finish, enter 'Y/N'
Y

Process finished with exit code 0
```

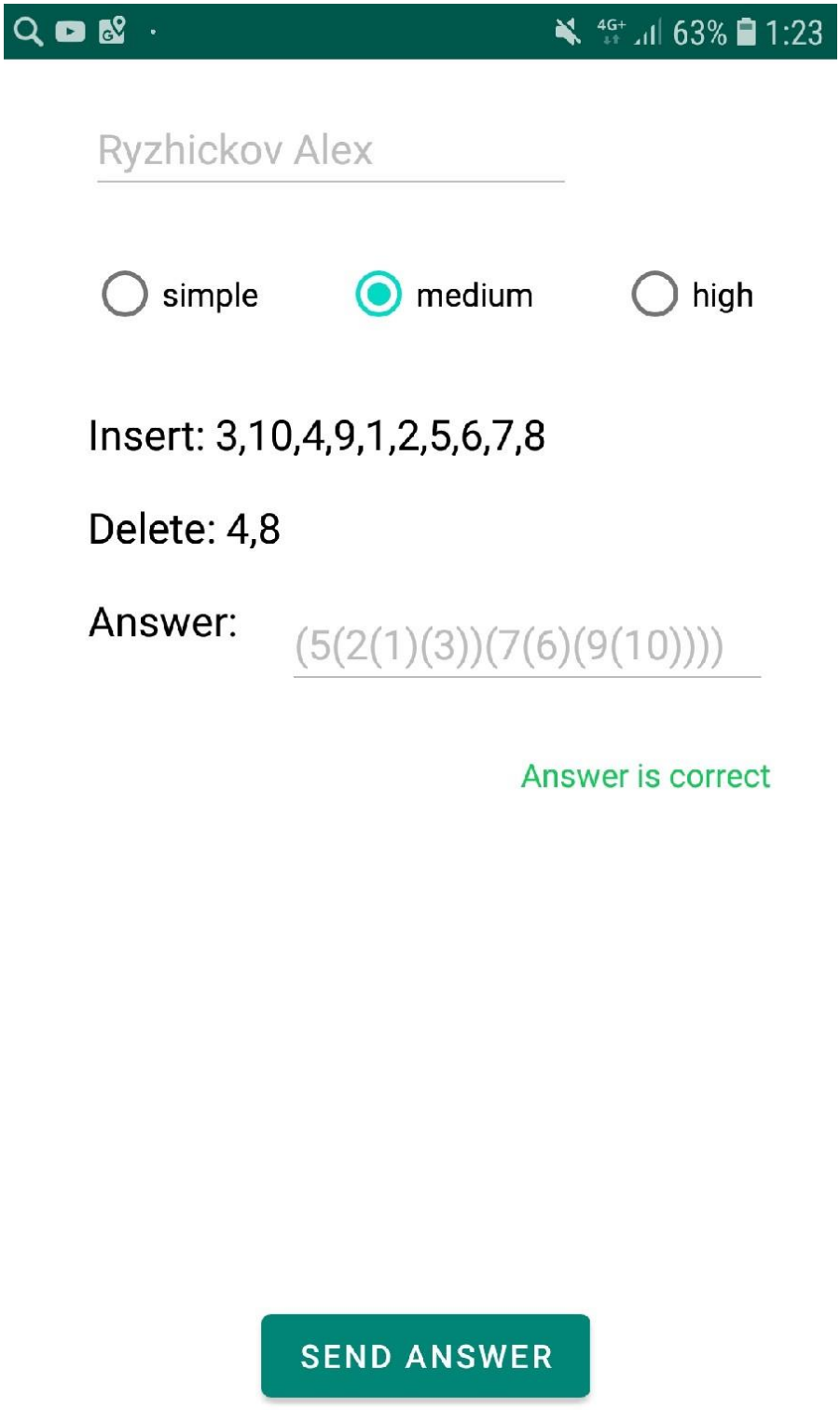
3. Программа выводит варианты заданий и ответы к ним

# 4 Android приложение

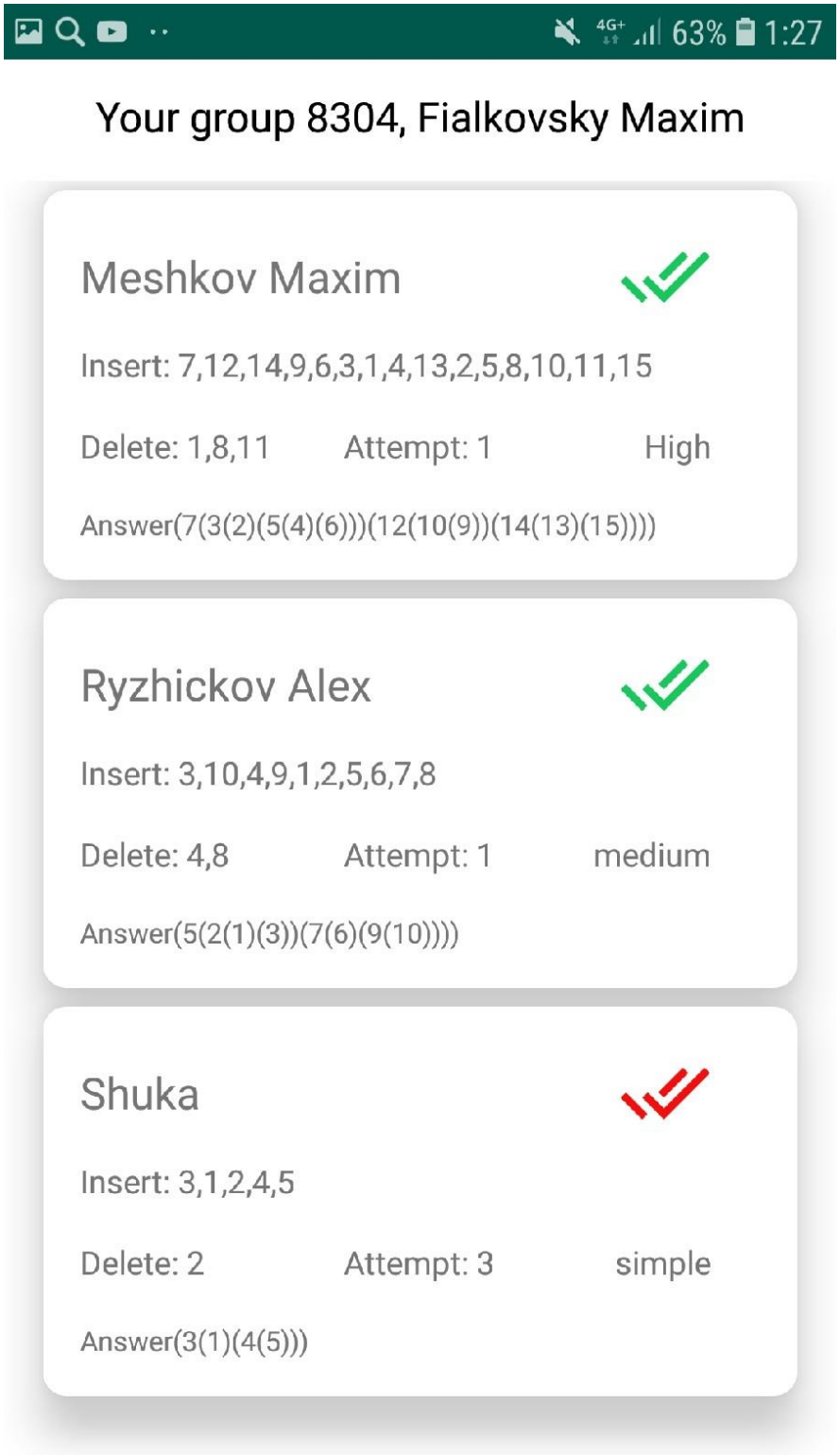
1. Выберите нужный вам вариант



2. Если вы выбрали кнопку студент, то введите ваше имя, выберите уровень сложности и нажмите кнопку “CONNECT”, после этого вам будет выдано задание. Необходимо ввести ответ (построенное вами дерево) в скобочной записи. У вас есть 3 попытки.



2. Если вы выбрали кнопку учитель, то введите ваше пароль (“makometr”)
Учителю доступен список учеников и информация об их вариантах :
сами задание, уровень их сложности, количество попыток и выполнено ли они.
Также учитель может удалять учеников из списка (свайпом влево или вправо).
Данные в списке представлены в реальном времени.



# ТЕСТЫ

## ВХОДНЫЕ ДАННЫЕ

1) **5 1 2 3 4 5 1 4**
2) **10 9 7 8 6 10 4 5 1 3 2 3 2 1 4**
3) **3 1 2 3 0**
4) **1 1 0**
5) **5 2 4 5 7 1 2 1 4**
6) **10 10 6 1 9 8 5 4 2 3 7 2 7 5**
7) **10 1 7 2 8 3 4 5 9 6 10 2 1 5**
8) **10 10 9 6 7 2 1 4 3 5 8 2 8 5**
9) **15 2 14 4 7 13 1 15 11 6 3 9 5 12 8 10 3 14 8 7**
10) **15 4 13 9 6 1 15 2 12 3 5 7 8 10 11 14 3 6 10 3**

## ВЫХОДНЫЕ ДАННЫЕ

**OPTION NUMBER 1**

**INSERT: 1 2 3 4 5**

**DELETE: 4**

**ANSWER: (2(1)(5(3)))**

**OPTION NUMBER 2**

**INSERT: 9 7 8 6 10 4 5 1 3 2**

**DELETE: 2 1 4**

**ANSWER: (8(5(3)(6(7)))(9(10)))**

**OPTION NUMBER 3**

**YOU ENTERED INCCORECT DELETE COUNT ELEMENTS**

**COUNT DELETE ELEMENTS = 0**

**INSERT: 1 2 3**

**DELETE:**

**ANSWER: (2(1)(3))**

**OPTION NUMBER 4**

**YOU ENTERED INCCORECT DELETE COUNT ELEMENTS**

**COUNT DELETE ELEMENTS = 0**

**INSERT: 1**

**DELETE:**

**ANSWER: (1)**

**OPTION NUMBER 5**

**INSERT: 2 4 5 7 1**

**DELETE: 1 4**

**ANSWER: (5(2)(7))**

**OPTION NUMBER 6**

**INSERT: 10 6 1 9 8 5 4 2 3 7**

**DELETE: 7 5**

**ANSWER: (6(2(1)(4(3)))(9(8)(10)))**

**OPTION NUMBER 7**

**INSERT: 1 7 2 8 3 4 5 9 6 10**

**DELETE: 1 5**

**ANSWER: (7(3(2)(6(4)))(9(8)(10)))**

**OPTION NUMBER 8**

**INSERT: 10 9 6 7 2 1 4 3 5 8**

**DELETE: 8 5**

**ANSWER: (6(2(1)(4(3)))(9(7)(10)))**

**OPTION NUMBER 9**

**INSERT: 2 14 4 7 13 1 15 11 6 3 9 5 12 8 10**

**DELETE: 14 8 7**

**ANSWER: (9(4(2(1)(3))(6(5)))(11(10)(13(12)(15))))**

**OPTION NUMBER 10**

**INSERT: 4 13 9 6 1 15 2 12 3 5 7 8 10 11 14**

**DELETE: 6 10 3**

**ANSWER: (7(4(2(1))(5))(13(9(8)(11(12)))(15(14))))**

# ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была создана программа на языке программирования C++, которая создаёт необходимое количество вариантов различной сложности и ответы к ним. Также было написано мобильное приложение под android на языке Java и с использованием базы данных Firebase, имеющие более гибкий функционал. Также были изучены AVL-деревья и получен опыт работы с ними.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Сайт ИТМО

https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%92%D0%9B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE

2)Firebase официальная документация

https://firebase.google.com/

3)JAVA И C / C ++: РУКОВОДСТВО ПО JNI

https://devarea.com/java-and-cc-jni-guide/#.Xe_4KugzaUm

# ПРИЛОЖЕНИЕ А

# НАЗВАНИЕ ПРИЛОЖЕНИЯ

```cpp
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <ctime>

const int right_overweight = 2;
const int left_overweight = -2;

struct Node // структура для представления узлов дерева
{
    int key;
    unsigned char height;
    Node *left;
    Node *right;

    Node(int k) {
        key = k;
        left = right = 0;
        height = 1;
    }
};

unsigned char height(Node *p) {
    return p ? p->height : 0;
}

int bFactor(Node *p) {
    return height(p->right) - height(p->left);
}

void fixHeight(Node *p) {
    unsigned char hl = height(p->left);
    unsigned char hr = height(p->right);
    p->height = (hl > hr ? hl : hr) + 1;
}

Node *rotateRight(Node *p) // правый поворот вокруг p
{
    Node *q = p->left;
    p->left = q->right;
    q->right = p;
    fixHeight(p);
    fixHeight(q);
    return q;
}

Node *rotateLeft(Node *q) // левый поворот вокруг q
{
    Node *p = q->right;
    q->right = p->left;
    p->left = q;
    fixHeight(q);
    fixHeight(p);
    return p;
}
```

```
Node *balance(Node *p) // балансировка узла p
{
    fixHeight(p);
    if (bFactor(p) == right_overweight) {
        if (bFactor(p->right) < 0)
            p->right = rotateRight(p->right);
        return rotateLeft(p);
    }
    if (bFactor(p) == left_overweight) {
        if (bFactor(p->left) > 0)
            p->left = rotateLeft(p->left);
        return rotateRight(p);
    }
    return p; // балансировка не нужна
}

Node *insert(Node *p, int k) // вставка ключа k в дерево с корнем p
{
    if (!p) return new Node(k);
    if (k < p->key)
        p->left = insert(p->left, k);
    else
        p->right = insert(p->right, k);
    return balance(p);
}

Node *findMin(Node *p) // поиск узла с минимальным ключом в дереве p
{
    return p->left ? findMin(p->left) : p;
}

Node *removeMin(Node *p) // удаление узла с минимальным ключом из дерева p
{
    if (p->left == 0)
        return p->right;
    p->left = removeMin(p->left);
    return balance(p);
}

Node *remove(Node *p, int k) // удаление ключа k из дерева p
{
    if (!p) return 0;
    if (k < p->key)
        p->left = remove(p->left, k);
    else if (k > p->key)
        p->right = remove(p->right, k);
    else //  k == p->key
    {
        Node *q = p->left;
        Node *r = p->right;
        delete p;
        if (!r) return q;
        Node *min = findMin(r);
        min->right = removeMin(r);
        min->left = q;
        return balance(min);
    }
    return balance(p);
}

void printAVLTree(Node *p) {
```

```
    std::cout << "(";
    std::cout << p->key;
    if (p->left) {
        //std::cout << "L";
        printAVLTree(p->left);
    }
    if (p->right) {
        //std::cout << "R";
        printAVLTree(p->right);
    }
    std::cout << ")";
}

std::string createString(Node *p) {
    std::string myString;
    myString += "(";
    myString += std::to_string(p->key);
    if (p->left) {
        myString += createString(p->left);
    }
    if (p->right) {
        myString += createString(p->right);
    }
    myString += ")";
    return myString;
}

void printAVLTree2(Node *p, int maxHeight) {

    for (int i = 0; i < maxHeight; ++i) {
        std::cout << " ";
    }

    std::cout << p->key;
    std::cout << "\n";

    if (p->left) {
        printAVLTree2(p->left, maxHeight + 1);
    }
    if (p->right) {
        printAVLTree2(p->right, maxHeight + 1);
    }

}

void printAVLTree3(Node *p, std::ofstream &ofs) {

    if (p->left) {
        printAVLTree3(p->left, ofs);
    }
    ofs << p->key;

    if (p->right) {
        printAVLTree3(p->right, ofs);
    }

}

void deleteTree(Node *p) {
    if (p->left) {
        deleteTree(p->left);
```

```
    }
    if (p->right) {
        deleteTree(p->right);
    }
    delete p;
}

int chooseComplexity() {
    int complexity = 0;

    std::cout << "Choose the difficulty\n";
    std::cout << "If you want to choose a simple difficulty - enter 1\n";
    std::cout << "If you want to choose a medium difficulty - enter 2\n";
    std::cout << "If you want to choose a high difficulty   - enter 3\n";
    std::cout << "If you want to choose a custom difficulty - enter 4\n";


    std::cin >> complexity;

    if ((complexity != 1) && (complexity != 2) && (complexity != 3) && (complexity !=
4)) {
        std::cout << "You entered incorrect complexity\n";
        return 0;
    }

    return complexity;


}


int countOptions() {
    int countOptions = 0;
    std::cout << "Enter the number of options (from 1 to 30)\n";

    std::cin >> countOptions;
    if (!(countOptions > 0 && countOptions <= 30)) {
        std::cout << "You entered incorrect options\n";
        return 0;
    }

    return countOptions;
}

int random(int min, int max) {
    return min + rand() % (max - min);
}


void createRandomArray(int *arrayNumbers, int *arrayOptions, int sizeArray) {
    for (int j = 0; j < sizeArray; ++j) {

        if (arrayNumbers[arrayOptions[j] - 1] != 0) {
            arrayNumbers[arrayOptions[j] - 1] = 0;
        } else {
            int min = 0;
            for (int i = 0; i < sizeArray; ++i) {
                if (arrayNumbers[i] != 0) {
                    min = arrayNumbers[i];
                    break;
                }
```

```
            }

            arrayOptions[j] = min;
            arrayNumbers[min - 1] = 0;

        }
    }
}

void printTask(int *arrayOptions, int *arrayDeleteElements, int sizeArray, int
countDeleteElements, std::ofstream &ofs) {
    std::cout << "      Insert: ";
    ofs << "      Insert: ";
    for (int i = 0; i < sizeArray; ++i) {
        std::cout << arrayOptions[i] << " ";
        ofs << arrayOptions[i] << " ";
    }
    std::cout << "\n      Delete: ";
    ofs << "\n      Delete: ";
    for (int j = 0; j < countDeleteElements; ++j) {
        std::cout << arrayDeleteElements[j] << " ";
        ofs << arrayDeleteElements[j] << " ";
    }
    std::cout << "\n";
    ofs << "\n";
}

void createVariant(int complexity, std::istream &fin, bool isReadFromUser,
std::ofstream &ofs) {
    int sizeArray = 0;
    int countDeleteElements = 0;

    if (complexity == 1) {
        sizeArray = 5;
        countDeleteElements = 1;
    }
    if (complexity == 2) {
        sizeArray = 10;
        countDeleteElements = 2;
    }
    if (complexity == 3) {
        sizeArray = 15;
        countDeleteElements = 3;
    }

    if (complexity == 4) {
        if (isReadFromUser) {
            std::cout << "Enter count elements:\n";
        }
        fin >> sizeArray;
        if (sizeArray > 0 && sizeArray <= 20) {
            countDeleteElements = sizeArray;
        } else {
            if(!isReadFromUser){
                return;
            }
            std::cout << "You entered inccorect count elements\n";
            std::cout << "Count elements = 20\n";
            sizeArray = 10;
            countDeleteElements = sizeArray;
        }
```

```
    }

    int *arrayNumbers = new int[sizeArray];
    int *arrayOptions = new int[sizeArray];
    int *arrayDeleteElements = new int[countDeleteElements];

    if (complexity != 4) {
        for (int i = 0; i < sizeArray; ++i) {
            arrayNumbers[i] = i + 1;
            arrayOptions[i] = random(1, sizeArray + 1);
        }

        createRandomArray(arrayNumbers, arrayOptions, sizeArray);

        for (int i = 0; i < countDeleteElements; ++i) {
            arrayDeleteElements[i] = arrayOptions[i];
            arrayDeleteElements[i] = arrayOptions[arrayDeleteElements[i] - 1];
        }
    } else {
        if (isReadFromUser) {
            std::cout << "Enter elements:\n";
        }
        for (int i = 0; i < sizeArray; ++i) {
            fin >> arrayOptions[i];
        }
        if (isReadFromUser) {
            std::cout << "Enter count delete elements: \n";
        }
        fin >> countDeleteElements;


        if (!(countDeleteElements > 0 && countDeleteElements <= sizeArray)) {
            std::cout << "You entered inccorect delete count elements\n";
            std::cout << "Count delete elements = 0\n";
            countDeleteElements = 0;
        }

        if (isReadFromUser) {
            std::cout << "Enter delete elements:\n";
        }

        for (int i = 0; i < countDeleteElements; ++i) {
            fin >> arrayDeleteElements[i];
        }
    }

    printTask(arrayOptions, arrayDeleteElements, sizeArray, countDeleteElements,ofs);

    auto *avlTree = new Node(arrayOptions[0]);
    for (int j = 1; j < sizeArray; ++j) {
        avlTree = insert(avlTree, arrayOptions[j]);
    }

    for (int k = 0; k < countDeleteElements; ++k) {
        avlTree = remove(avlTree, arrayDeleteElements[k]);
    }

    std::cout << "      Answer: ";
    ofs << "      Answer: ";
    if (avlTree != 0) {
        std::cout << createString(avlTree);
```

```
        ofs << createString(avlTree);
    } else {
        std::cout << "Empty tree";
        ofs << "Empty tree";
    }
    std::cout << "\n";
    ofs << "\n";
    if (avlTree != 0) {
        deleteTree(avlTree);
    }
    delete[] arrayNumbers;
    delete[] arrayOptions;
    delete[] arrayDeleteElements;

}

int main() {

    srand(time(nullptr));
    std::ofstream ofs("C:\\Users\\Alex\\Desktop\\output.txt", std::ios_base::app);


    std::cout << "Welcome to the verification program\n"
                 "The program creates variants of different and answers\n\n";

    std::cout << "If you want to see tests entre 'test', otherwise just any
characters\n";

    std::string test;

    std::cin >> test;

    if (test == "test") {
        std::ifstream fin;
        fin.open("C:\\Users\\Alex\\Desktop\\test5.txt");

        if (fin.is_open()) {
            std::cout << "Reading from file:" << "\n";

            int super_count = 0;

            while (!fin.eof()) {

                std::cout << "Option number " << super_count + 1 << "\n";
                ofs << "Option number " << super_count + 1 << "\n";
                createVariant(4, fin, false,ofs);
                super_count++;

            }
        } else {
            std::cout << "File not opened";
        }

        fin.close();
    }

    int count = 0;
    int complexity = 0;
    while (true) {
        complexity = chooseComplexity();
        if (complexity == 0) {
```

```
            break;
        }
        if (complexity != 4) {
            count = countOptions();
            if (count == 0) {
                break;
            }
        } else {
            count = 1;
        }
        for (int i = 0; i < count; ++i) {
            std::cout << "Option number " << i + 1 << "\n";
            ofs << "Option number " << i + 1 << "\n";
            createVariant(complexity, std::cin, true,ofs);
        }
        char choice;
        std::cout << "If you want to finish, enter 'Y/N'\n";
        std::cin >> choice;
        if (choice == 'Y' || choice == 'y') {
            break;
        }
    }

    ofs.close();

    return 0;
}
```